IT & C

ISSN 2821 - 8469, ISSN – L 2821 - 8469, Volumul 2, Numărul 2, Iunie 2023

# Imaginea memoriei în programarea cu Python

Nicolae Sfetcu

Sfetcu, Nicolae (2023), Imaginea memoriei în programarea cu Python, *IT & C*, **2**:2, 33-38, DOI: 10.58679/IT13240, https://www.internetmobile.ro/imaginea-memoriei-in-programarea-cu-python/

Publicat online: 07.05.2023

# Imaginea memoriei în programarea cu Python

Nicolae Sfetcu
nicolae@sfetcu.com

**Memory Image in Python Programming**

**Abstract**

Python is a multi-paradigm programming language that pursues a simpler, less cluttered syntax and grammar while giving developers a choice in their coding methodology. In most situations, it is recommended to allocate memory from the Python heap, since the latter is under the control of the Python memory manager. Even when the requested memory is used solely for internal, very specific purposes, delegating all memory requests to the Python memory manager gives the interpreter a more accurate picture of its overall memory footprint. For learning the Python programming language, it is easy to work with memory images in a very specific format.

**Keywords**: programming, programming language, Python, memory, memory image

**Rezumat**

Python este un limbaj de programare multi-paradigmă care urmărește o sintaxă și o gramatică mai simple, mai puțin aglomerate, oferind în același timp dezvoltatorilor o alegere în metodologia lor de codare. În cele mai multe situații, se recomandă alocarea memoriei din heap-ul Python, deoarece acesta din urmă este sub controlul managerului de memorie Python. Chiar și atunci când memoria solicitată este utilizată exclusiv în scopuri interne, foarte specifice, delegarea tuturor solicitărilor de memorie către managerul de memorie Python face ca interpretul să aibă o imagine mai precisă a amprentei sale de memorie în ansamblu. Pentru învățarea limbajului de programare Python, este ușor de lucrat cu imaginile memoriei, într-un format foarte specific.

**Cuvinte cheie**: programare, limbaj de programare, Python, memorie, imaginea memoriei

IT & C, Volumul 2, Numărul 2, Iunie 2023, pp. 33-38
ISSN 2821 - 8469, ISSN – L 2821 – 8469, DOI: 10.58679/IT13240
URL: https://www.internetmobile.ro/imaginea-memoriei-in-programarea-cu-python/

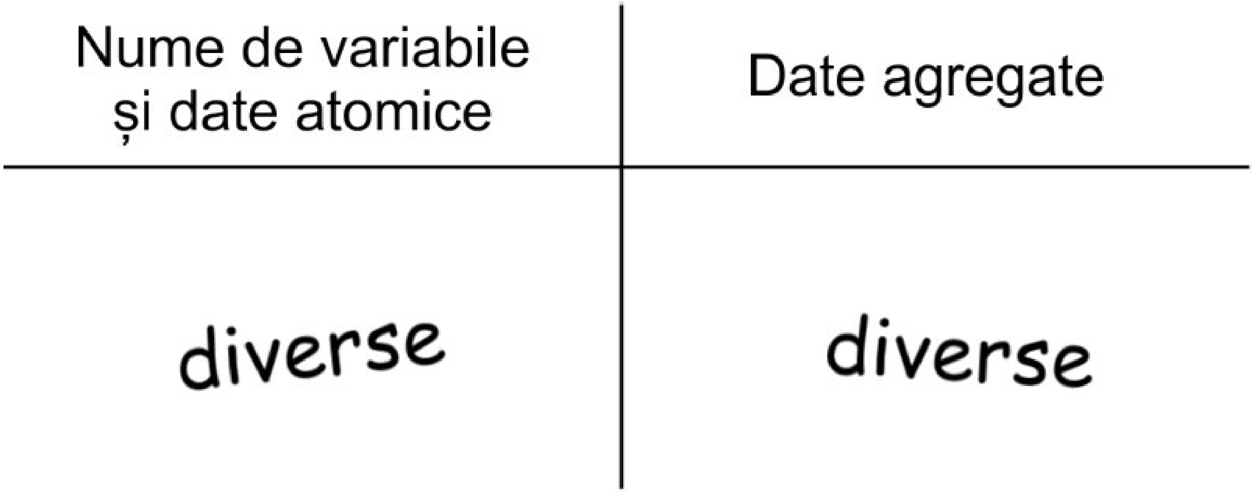
## Introducere

Python urmărește o sintaxă și o gramatică mai simple, mai puțin aglomerate, oferind în același timp dezvoltatorilor o alegere în metodologia lor de codare. Este un limbaj de programare multi-paradigmă. Programarea orientată pe obiecte și programarea structurată sunt pe deplin acceptate, iar multe dintre caracteristicile sale acceptă programarea funcțională și programarea orientată pe aspect (inclusiv prin metaprogramare și metaobiecte (metode magice)). Multe alte paradigme sunt acceptate prin intermediul extensiilor, inclusiv proiectarea prin contract și programarea logică.

În cele mai multe situații, se recomandă alocarea memoriei din heap-ul Python, deoarece acesta din urmă este sub controlul managerului de memorie Python. De exemplu, acest lucru este necesar atunci când interpretul este extins cu noi tipuri de obiecte scrise în C. Un alt motiv pentru utilizarea heap-ului Python este dorința de a informa managerul de memorie Python despre nevoile de memorie ale modulului de extensie. Chiar și atunci când memoria solicitată este utilizată exclusiv în scopuri interne, foarte specifice, delegarea tuturor solicitărilor de memorie către managerul de memorie Python face ca interpretul să aibă o imagine mai precisă a amprentei sale de memorie în ansamblu.

Pentru învățarea limbajului de programare Python, este ușor de lucrat cu imaginile memoriei. **Imaginile memoriei** vor avea un format foarte specific, iar acest lucru este crucial: nu fiți creativi cu modul în care sunt etichetate lucrurile sau unde sunt desenate lucrurile. Pentru ca să funcționeze codul dvs., *trebuie să aveți această imagine exact corectă*. Nu este artă; este știință.

Imaginile noastre ale memoriei vor fi întotdeauna împărțite în exact două „zone”, una în stânga și una în dreapta, etichetate după cum urmează:

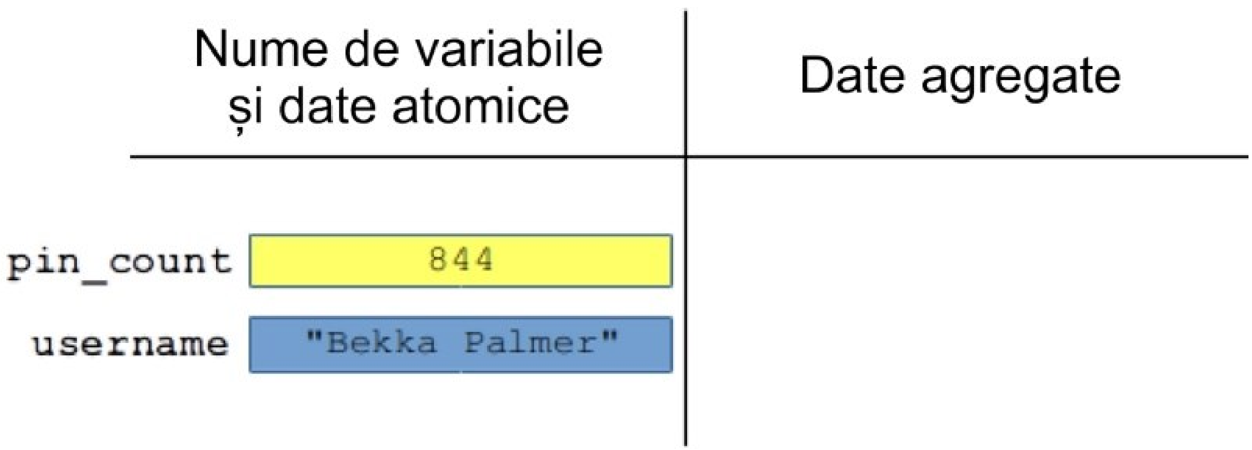



**Scrierea în memorie**

Când creăm variabile atomice într-o celulă Code, precum:

```
pin_count = 844
username = 'Bekka Palmer'
```

fiecare este pusă pe partea stângă a diagramei ca o **casetă cu nume**. Numele casetei este numele variabilei, iar elementul din casetă este valoarea acesteia.

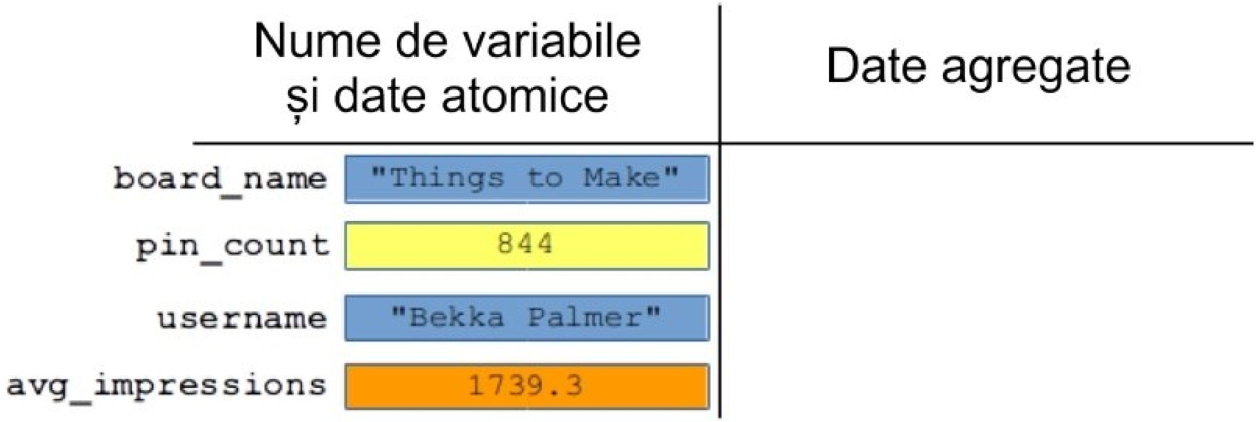



Nu contează ce casete sunt mai mari sau mai mici pe pagină, doar că fiecare nume se asociază cu caseta corespunzătoare și nu se încurcă. Ca bonus, am colorat diferit casetele, indicând faptul că pin_count (int) este un alt tip decât numele de utilizator (str)[1].

[1] Un alt detaliu minuscul pe care l-ați putea observa: chiar dacă codul nostru avea ghilimele simple pentru a delimita numele Bekka Palmer, am pus ghilimele duble în caseta din imaginea memoriei. Acest lucru subliniază faptul că, indiferent de modul în care creați un șir în cod – indiferent dacă este vorba de ghilimele simple sau duble – „lucrul" care se scrie în memorie este același. De fapt, cele stocate sunt de fapt caracterele Bekka Palmer fără ghilimele. Totuși, îmi place să introduc ghilimele în imaginile memoriei, doar pentru a sublinia natura șirului valorii.

Crearea mai multor variabile adaugă doar mai multe casete cu nume:

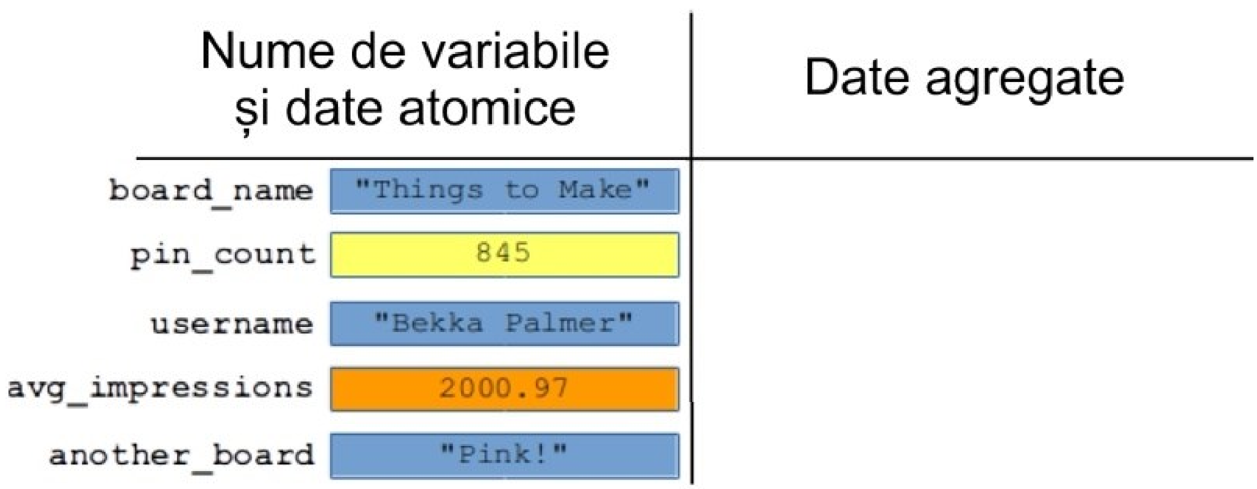
```
...
avg_num_impressions = 1739.3
board_name = "Things to Make"
```




Am amestecat în mod deliberat ordinea casetelor. Python nu oferă nicio garanție cu privire la „ordinea" în care va stoca variabilele. Garanțiile Python sunt doar că va stoca în mod constant un nume, o valoare și un tip pentru fiecare variabilă.

Când schimbăm valoarea unei variabile (mai degrabă decât să creăm una nouă), valoarea din caseta corespunzătoare se actualizează:

```
…
avg_num_impressions = 2000.97
pin_count = 845
another_board = 'Pink!'
```



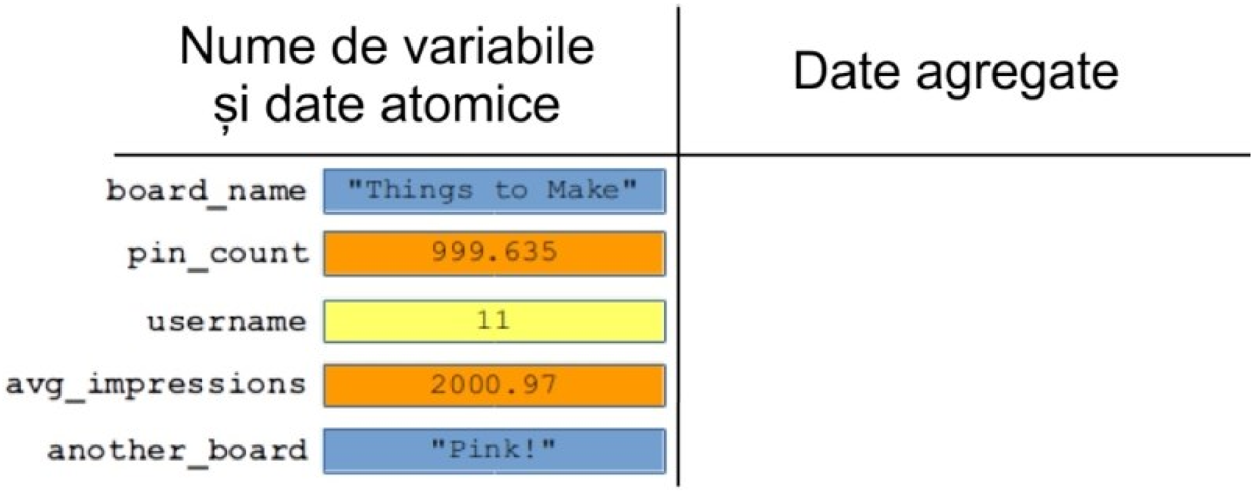

Rețineți că *valoarea anterioară din casetă este complet ștearsă* și nu există absolut nicio modalitate de a o recupera vreodată. De fapt, nu există nicio modalitate de a ști că a existat chiar

o valoare anterioară diferită de cea actuală. Cu excepția cazului în care sunt organizate în mod specific, programele de computer țin doar evidența prezentului, nu a trecutului.

Un alt aspect: spre deosebire de unele limbaje de programare (așa-numitele limbaje „puternic tipizate”, precum Java sau C++), chiar și *tipul* valorii pe care o deține o variabilă se poate schimba dacă doriți. Chiar dacă următorul exemplu nu are prea mult sens, să presupunem că am scris acest cod în continuare:

```
…
pin_count = 999.635
username =11
```

Acest lucru face ca nu numai conținutul casetelor să se schimbe, ci chiar și culorile acestora. Variabila username a fost acum un moment, dar acum este int.




**Citirea din memorie**

„Citirea din memorie” înseamnă doar referirea la o variabilă pentru a-i recupera valoarea. Până în prezent, nu știm cum să facem cu excepția tipăririi:

```
print("The {} board has {} pins.".format(another_board,
    pin_count))
```

```
The Pink! board has 999.635 pins.
```

Important este că imaginea memoriei este (doar) înregistrarea curentă și fiabilă a aspectului memoriei în orice punct al unui program. Gândiți-vă la asta ca reflectând un instantaneu în timp: imediat după executarea unei linii de cod – și chiar înainte de următoarea – putem consulta imaginea pentru a obține valoarea fiecărei variabile. Exact asta face, practic, Python.

Subliniez acest punct, deoarece există mulți începători care privesc codul complicat și încearcă să „gândească” ce valoare va avea fiecare variabilă pe măsură ce rulează. Este greu de făcut cdoar cu câteva linii. Pentru a ține evidența a ceea ce s-a schimbat în ceea ce s-a schimbat și când, trebuie să păstrați cu adevărat o listă actualizată a valorii fiecărei variabile pe măsură ce programul se execută … ceea ce este de fapt exact ceea ce este imaginea memoriei.

Deci, dacă încercați să vă dați seama „ce va ieși din acest program dacă imprim variabila odometer imediat după linia 12?” nu încercați să priviți codul și să-i reconstruiți comportamentul de la zero. În schimb, desenați o imagine a memoriei, actualizați-o în mod corespunzător în timp ce parcurgeți fiecare linie de cod, apoi uitați-vă la ea pentru răspuns.

## Bibliografie

Stephen Davies, *The Crystal Ball – Instruction Manual*, Vol. 1: Introduction to Data Science, v. 1.1. Copyright © 2021 Stephen Davies. Licența CC BY-SA 4.0. Traducere și adaptare: Nicolae Sfetcu. © 2021 MultiMedia Publishing, *Introducere în Știința Datelor*, Volumul 1